

# Next Token Generation

---

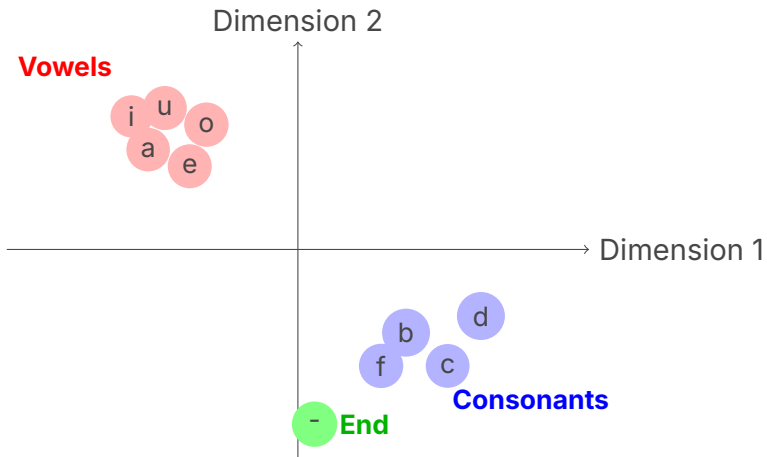
Nipun Batra

IIT Gandhinagar

July 30, 2025

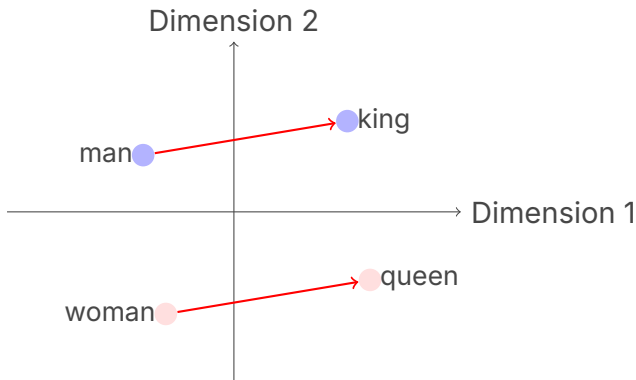
**Vocabulary Size:**

26 letters + 1 hyphen = **27 characters**



# Word2Vec Analogy Example

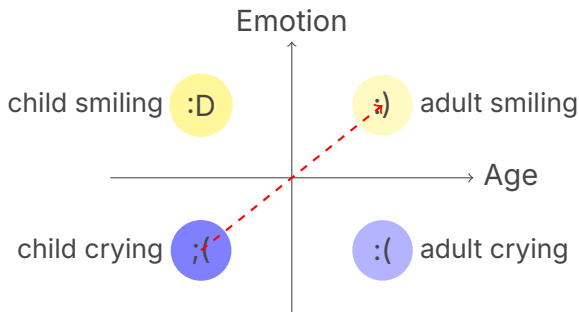
## Classic Word2Vec Relationship



**Relationship:**  $\text{queen} \approx \text{king} - \text{man} + \text{woman}$

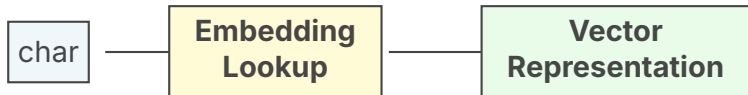
# Analogy with Emotions

## Emotional Expression Analogy



**Relationship:** child crying = child smiling + adult crying - adult smiling

# Embedding Matrix/Table Concept



**Process:** Character → Lookup in Embedding Table → Dense Vector

# Embedding Table Structure

**27 × K Embedding Matrix**

Char	D1	D2	...	DK
a	0.2	-0.1	...	0.8
b	-0.3	0.5	...	-0.2
c	0.1	0.3	...	0.4
⋮	⋮	⋮	⋮	⋮
z	0.7	-0.4	...	0.1
-	0.0	0.9	...	-0.5

## Key Point

Each character maps to a K-dimensional vector.

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters



# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random
  - Updated during training via backpropagation

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
  - Transform concatenated embeddings to output

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
  - Transform concatenated embeddings to output
  - Learn classification patterns

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
  - Transform concatenated embeddings to output
  - Learn classification patterns
- **Total Learnable Parameters:**



# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
  - Transform concatenated embeddings to output
  - Learn classification patterns
- **Total Learnable Parameters:**
  - Embedding:  $27 \times K$

# Learnable Parameters

- **Embedding Matrix:**  $27 \times K$  parameters
  - Initially random
  - Updated during training via backpropagation
  - Learns meaningful character representations
- **Neural Network Weights:** MLP parameters
  - Transform concatenated embeddings to output
  - Learn classification patterns
- **Total Learnable Parameters:**
  - Embedding:  $27 \times K$
  - MLP:  $(\text{context\_size} \times K) \rightarrow \text{hidden} \rightarrow \dots \rightarrow 27$

# Example: 2D Embeddings for "abi"

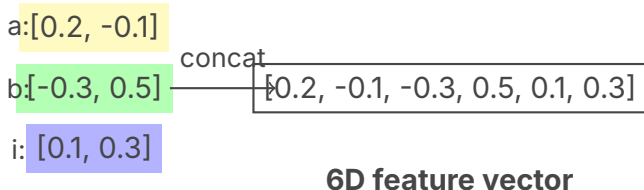
## Embedding Matrix (27 × 2)

Input:  $X = ["a", "b", "i"]$

	D1	D2	
a	0.2	-0.1	[0.2, -0.1]
b	-0.3	0.5	[-0.3, 0.5]
...	...	...	
i	0.1	0.3	[0.1, 0.3]
...	...	...	
z	0.7	-0.4	
-	0.0	0.9	

# Concatenate the Embeddings

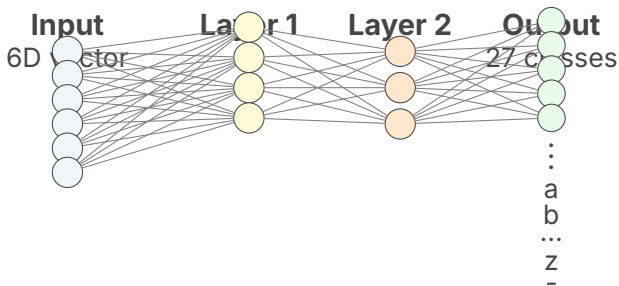
## Feature Vector Construction



## Result

3 chars  $\times$  2D embeddings = 6D input to neural network

# Multi-Layer Perceptron Architecture



# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**



# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input  $\rightarrow$  Embeddings  $\rightarrow$  Concatenate  $\rightarrow$  MLP  $\rightarrow$  Probabilities



# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input  $\rightarrow$  Embeddings  $\rightarrow$  Concatenate  $\rightarrow$  MLP  $\rightarrow$  Probabilities
  2. Compute cross-entropy loss

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input  $\rightarrow$  Embeddings  $\rightarrow$  Concatenate  $\rightarrow$  MLP  $\rightarrow$  Probabilities
  2. Compute cross-entropy loss
  3. Backward pass: Update both embeddings and MLP weights

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input  $\rightarrow$  Embeddings  $\rightarrow$  Concatenate  $\rightarrow$  MLP  $\rightarrow$  Probabilities
  2. Compute cross-entropy loss
  3. Backward pass: Update both embeddings and MLP weights

# Training Objective

- **Loss Function:** Cross-entropy loss for multi-class classification

$$\mathcal{L} = - \sum_{i=1}^N \sum_{c=1}^{27} y_{i,c} \log(\hat{y}_{i,c}) \quad (1)$$

- **What we learn:**
  1. **Embedding Matrix:** Character representations ( $27 \times K$  parameters)
  2. **MLP Weights:** Neural network parameters for classification
- **Training Process:**
  1. Forward pass: Input  $\rightarrow$  Embeddings  $\rightarrow$  Concatenate  $\rightarrow$  MLP  $\rightarrow$  Probabilities
  2. Compute cross-entropy loss
  3. Backward pass: Update both embeddings and MLP weights
  4. Repeat for all training examples

# Sampling from the Learned Model

**Test Input:** "abi"

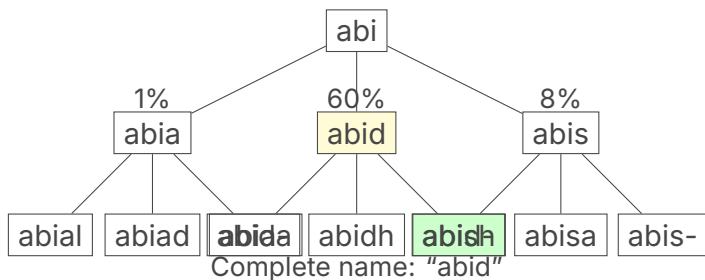
## Predicted Probability Distribution

Next Char	Probability	Next Char	Probability
a	0.01	n	0.05
b	0.01	o	0.02
c	0.03	p	0.01
d	<b>0.60</b>	q	0.00
e	0.02	r	0.03
f	0.01	s	0.08
...	...	...	...
-	0.05	z	0.01

**Most Likely Continuation**

"abi" → "abid" (60

# Generation Tree Structure



**Recursive Process:** Sample next character, append, repeat until end token

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$



# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \quad (3)$$

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \quad (3)$$

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \quad (3)$$

- **Temperature Effects:**

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \quad (3)$$

- **Temperature Effects:**

- $T = 1$ : Standard probabilities

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \quad (3)$$

- **Temperature Effects:**

- $T = 1$ : Standard probabilities
- $T \rightarrow 0$ : More peaked (deterministic)

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \quad (3)$$

- **Temperature Effects:**

- $T = 1$ : Standard probabilities
- $T \rightarrow 0$ : More peaked (deterministic)

# Temperature in Softmax

- **Standard Softmax:**

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{27} e^{z_j}} \quad (2)$$

- **Temperature-scaled Softmax:**

$$P(y_i) = \frac{e^{z_i/T}}{\sum_{j=1}^{27} e^{z_j/T}} \quad (3)$$

- **Temperature Effects:**

- $T = 1$ : Standard probabilities
- $T \rightarrow 0$ : More peaked (deterministic)
- $T \rightarrow \infty$ : More uniform (random)

# Temperature Variations

**Context:** "abi" → Next character probabilities

Char	T=0.5 (Low)	T=1.0 (Default)	T=2.0 (High)
a	0.001	0.01	0.08
d	<b>0.95</b>	<b>0.60</b>	<b>0.25</b>
s	0.01	0.08	0.12
h	0.005	0.03	0.09
-	0.02	0.05	0.11
others	0.015	0.23	0.35

- **Low T:** Conservative, predictable



# Temperature Variations

**Context:** "abi" → Next character probabilities

Char	T=0.5 (Low)	T=1.0 (Default)	T=2.0 (High)
a	0.001	0.01	0.08
d	<b>0.95</b>	<b>0.60</b>	<b>0.25</b>
s	0.01	0.08	0.12
h	0.005	0.03	0.09
-	0.02	0.05	0.11
others	0.015	0.23	0.35

- **Low T:** Conservative, predictable

# Temperature Variations

**Context:** "abi" → Next character probabilities

Char	T=0.5 (Low)	T=1.0 (Default)	T=2.0 (High)
a	0.001	0.01	0.08
d	<b>0.95</b>	<b>0.60</b>	<b>0.25</b>
s	0.01	0.08	0.12
h	0.005	0.03	0.09
-	0.02	0.05	0.11
others	0.015	0.23	0.35

- **Low T:** Conservative, predictable
- **High T:** Creative, diverse

# Key Takeaways

- **Core Idea:** Next token prediction as classification

# Key Takeaways

- **Core Idea:** Next token prediction as classification

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling



# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
  - GPT models use the same principle

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
  - GPT models use the same principle
  - Scaled to words/subwords instead of characters

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
  - GPT models use the same principle
  - Scaled to words/subwords instead of characters



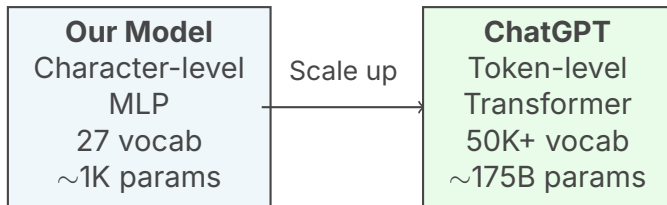
# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
  - GPT models use the same principle
  - Scaled to words/subwords instead of characters
  - Transformer architecture instead of MLP

# Key Takeaways

- **Core Idea:** Next token prediction as classification
- **Representation Learning:** Character embeddings capture similarity
- **Architecture:** Embeddings + MLP for sequence modeling
- **Training:** Joint learning of embeddings and classifier weights
- **Generation:** Autoregressive sampling with temperature control
- **Applications:** Foundation for modern language models
  - GPT models use the same principle
  - Scaled to words/subwords instead of characters
  - Transformer architecture instead of MLP
  - Billions of parameters instead of thousands

# From Character-Level to ChatGPT



**Same fundamental principle: Predict the next token!**